More Efficient Server Assisted One Time Signatures

Vipul Goyal
Department of Computer Science & Engineering
Institute of Technology
Banaras Hindu University
Varanasi, India
Vipul.goyal@cse04.itbhu.org

Abstract

Server assisted one time signature scheme was recently presented as a non-repudiation service for mobile and constrained devices. However, the scheme suffered with high storage requirements for the virtual server and high memory requirements for the mobile client. We improve the scheme by significantly reducing virtual server storage requirements as well as mobile client memory requirements. More precisely, the virtual server storage requirements in our scheme are reduced by a factor of more than 80 compared to the original scheme. Further, memory requirements for the mobile client are reduced by a factor of more than 130. This is done by generating various quantities pseudorandomly and storing just their cryptographic hash (instead of storing them fully) wherever possible, while still being able to perform dispute resolution.

1 Introduction

Electronic transactions are a reality in today's world. To have some legal significance, these transactions should have some form of non-repudiation [7]. This is usually provided through a digital signature. However, digital signature generation and even verification are known to be computationally intensive processes. It is not always practical to implement public key cryptography and hence digital signatures on a mobile device having limited computational resources and memory. Hence the question arises: "how to provide a means of non-repudiation to such devices on which public key cryptography is not always practical?" Answering this question is very important in order realize the full potential of pervasive computing.

One answer is to employ a third party. If unconditionally trusted, very efficient non-repudiation services may be provided by the third party. In a straightforward setting, the mobile user, also called the originator, and the trusted third party may share a secret key. The originator would supply the message to be signed encrypted with that key and the third party would sign that message on behalf of the originator. Clearly, it is easy for the third party to cheat in this setting since it could sign any message on behalf of the user.

Such schemes are of limited usage since they require the originator to have unconditional trust in the third party. More practical, though less efficient, techniques are possible in which if the third party cheats, the originator is able to prove this cheating to an arbiter. The third party in this setting is called a verifiable server (VS). The first practical scheme under this category was devised by Asokan et al [7]. However, it suffered from several limitations as discussed in section 3. This scheme was recently improved by Bicakci and Baykal [1] to propose server assisted one time signatures (SAOTS). SAOTS, although practical, suffered from storage problems. The storage requirements for the VS in this scheme were quite high and ever increasing. More precisely, for every signature generated, the VS was required to add approximately 5 KB to its storage. Assuming that a user signs just a single message per day and the VS supports just ten thousand users, the storage requirements for the VS start becoming prohibitive in just 1 year. Clearly, it is not desirable to use this scheme on a large scale in a commercial setting. Additionally, the mobile device was required to store about 2.7 KB of secret data in its memory in order to be able to use this scheme. For a device which cannot generate or verify signatures, storing this data in its memory could also be prohibitive.

OUR CONTRIBUTION. We improve the SAOTS construction trying to decrease the storage requirements for the VS as well as the memory requirements for the mobile client. By generating one time signature keys pseudorandomly and introducing a new server storage and dispute resolution scheme, we are able to substantially reduce the VS storage requirements as well as the mobile client memory requirements. Only 60 bytes per signature generation are added to the VS storage instead of 5 KB in the previous scheme. Further, the memory requirements for the mobile client are reduced from 2.7 KB to just 20 bytes. The computational requirements in our scheme remain mostly the same as in the previous scheme [1].

Rest of the paper is organized as follows. Section 2 provides some background on one time signatures. Section 3 discusses the related work. Section 4 discusses the proposed construction of the SAOTS scheme and various issues involved in it. Section 5 concludes the paper.

2 Background on One Time Signatures

The Concept of One time signatures (OTS) has been known for over two decades. It was initially proposed by Lamport and was the first digital signatures scheme ever designed. Interestingly, OTS schemes employ nothing more than OWHFs. The concept of OTS was subsequently enhanced by Merkle [17-18], Winternitz [17] and Bicakci et al [22]. Bleichenbacher et al [19-21] formalized the concept of OTS using directed acyclic graphs (DAGs).

Let h be a one way function. To sign a one bit message [16], the signer chooses as the secret key two values x_1 and x_2 (representing '0' and '1') and publishes their images under the one-way function, $y_1 = h(x_1)$ and $y_2 = h(x_2)$, as the public key. These x's and y's are called the secret key components and the public key components, respectively. To sign a single bit, reveal the pre-image corresponding to the actual '0' or '1'. That is, reveal either x_1 or x_2 based upon whether the message to be signed is 0 or 1. For signing longer messages, several instances of this basic scheme may be used. Thus we note that to sign an n bit message, 2n x's and 2n y's are required. This means that the size of signatures generated is equal to n x's, i.e., n times the size of the secret values.

There are several improvements to this basic scheme. Most notably, Merkle [17, 18] proposed an improvement which reduces the number of public and secret key components as well as the signature size in the Lamport method by almost a factor of two.

3 Related Works

Server assisted signatures could come in 3 varieties depending upon the trusted placed in the server. It is possible to obtain perfectly elegant schemes if the server is fully trusted. However for obvious reasons, these schemes are useful only in very limited environments. On the other extreme, the server could be completely untrusted. That is, it would assist the client only in non-security sensitive computations. As noted by Bicakci et al [1], it turns out to be rather hard to design useful schemes utilizing a completely untrusted server. Indeed, most of the schemes suggested in this category have been broken. However, for DSA, a secure and still unbroken scheme was proposed by Jakobson and Wetzel [6]. They however compromise on efficiency. That is in their approach, to generate the signature, public key operations although in reduced amount are still needed to be performed on the constrained device [1].

The last alternative is to employ a semi-trusted server also called a verifiable server (VS). A VS can cheat at will, but then the user would have the ability to prove the cheating in a court of law. Thus, the protocols are designed so as to leave the user with a cryptographic proof against the VS in case it cheats.

The first work in the category is the SAS protocol [7, 8]. Here, the basic idea is to employ hash chains [15] to gain efficiency. The server signs a message initially which the client verifies and releases a link of the hash chain to certify its validity. To ensure that the server is not signing multiple messages per hash chain link, the client should store all the signatures ever generated by the server on its behalf. Bicakci et al [1] observe several serious limitations of the SAS protocol. Firstly, for every signature request, the client is required to perform a signature verification which could be expensive (especially when employing schemes like DSS). Secondly, as noted above, the constrained client should store all the signatures VS generated on its behalf which may be impractical. Finally, SAS signatures are not 'standard' signatures and they are not compatible with existing applications.

SAS was recently improved by Bicakci and Baykal [1] to propose server assisted one time signatures (SAOTS). SAOTS is the first VS based approach where the user does not need to perform any public key operation at all. It eliminates all the aforementioned problems with SAS. That is, it is compatible with existing applications, doesn't require the client to store all the signatures and even reduces the number of rounds required from 3 to 2. The basic idea is that the user signs the message with a one time signature key pair and sends it to the VS. The VS in turn stores the user's one time signature and signs the message with the traditional public key. More details follow.

For system setup, every user registers to the VS and generates a one-time key pair by randomly generating secret key components. In a secure fashion, the user distributes the public key to the server. For getting a message signed by the VS, the user precomputes a second one-time key pair. When the message to be signed is ready, he concatenates the message with the new one-time public key and signs this by his previous one-time private key. He then sends the message and the new one-time public key as well as the one-time signature to VS. VS verifies the received one-time signature using the one time public key received in the previous step. He stores the new one-time public key the user has signed for the verification of next message. VS also stores the received signature. He is now ready to sign the message with the user's private key. Finally, the signed message is transmitted to the intended receiver(s).

The user can sign any further messages easily by repeating the above steps. Dispute resolution is straightforward since VS stores all the public keys and signatures received form the user.

The main problems in this scheme are the high storage requirements for the VS and memory requirements for the user. Recall that for SHS and Merkle's construction of one time signatures, the number of secret (or public) key components = 160 + log(160) = 168. For every signature, the VS is required to store: the message to be signed = 20 bytes, the public key = 168*20 = 3360 bytes and the signature = (168/2)*20 = 1680 bytes. Hence, the total storage increase for every signature is 20 + 3360 + 1680 = 5060 bytes or approximately 5 KB. As explained in Section 1, even for the minimal parameters, the storage requirements for the VS start becoming prohibitive soon in such a setting. Further, the user is required to remember the last secret key components in order to be able to sign the next message. For 128 bit random numbers, the user memory requirements comes out to be approximately 168*16 = 2.7 KB. This can also be prohibitive for a device which cannot even generate or verify signatures.

4 The Proposed Construction

Before going further, we introduce some basic notations used in this section:

U	The mobile user or the originator
VS	Verifiable Server
R	Receiver of the signature
L	Length of the output of OWHF employed, e.g., 160 bit for SHS
m	Number of public/secret key components used in the OTS scheme. Equal to $L+\log_2(L)$ for Merkle's construction
p	Average number of components in a one time signature. Usually equal to m/2.
P_U^i	ith one time public key of user U. Equal to the collection (or Concatenation) of m public key components
S_U^i	ith one time secret key of user U. Equal to the collection (or Concatenation) of m secret key components
$S_U^i(M)$	The message M signed with the one time secret key S_U^i. Equal to the collection (or Concatenation) of the relevant secret key components required to sign M
P_U	Traditional public key of the user U
$S_U(M)$	Message M signed with the traditional secret key of the user U

4.1 Setup

In order to initialize the system, the mobile user U initiates a counter i = 1 and generates the following-

1) A secret key K
2) A one time key pair as follows-
   $S_U^i = \{h(K, i, 1), h(K, i, 2), \dots , h(K, i, m)\}$
   $P_U^i = \{h^2(K, i, 1), h^2(K, i, 2), \dots , h^2(K, i, m)\}$

Now, the user securely stores K and i and transfers P_U^1 to the VS in a non-repudiable manner.

$$\mathbf{U \rightarrow VS:} \qquad \mathbf{P_U^1}$$

In addition, to produce traditional public key signatures on behalf of the user, the VS generates a public/secret key pair i.e. P_U/S_U on behalf of the user and obtains a certificate from the CA specifying the public key P_U. Note that the secret key S_U is not revealed even to U itself.

4.2 Operation

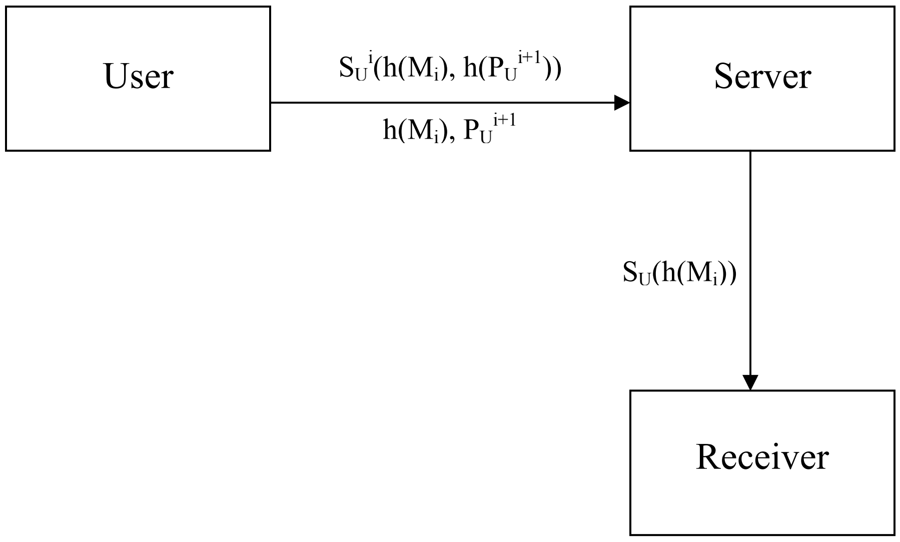

Figure 1: Operation of the Proposed Scheme

For generating the ith signature, the protocol works as follows-

1) The user precomputes the next i.e. (i+1)th one time public/secret key pair (see section 4.1). When the message to be signed is ready, he concatenates the hash of the message with the hash of the computed one-time public key and signs the resulting quantity with his current i.e. ith one time secret key. He then sends this signature along with the hash of the message and the computed public key to VS.

$$\mathbf{U \rightarrow VS:} \quad \mathbf{S_U^i(h(M_i), h(P_U^{i+1})), h(M_i), P_U^{i+1}}$$

2) VS checks the validity of the received signature using the stored P_U^i. It then stores the following-
   a) $h(M_i)$
   b) $h(S_U^i(h(M_i), h(P_U^{i+1})))$
   c) $h(P_U^i)$

   VS then replaces the stored P_U^i with the received P_U^{i+1}. Now, VS produces a traditional public key digital signature to sign the message $h(M_i)$ using user's secret key S_U. This signature is then directly sent to the receiver R if specified in the user's request. Alternatively, the signature may also be returned to the user.

$$\mathbf{VS \rightarrow R:} \quad \mathbf{S_U(h(M_i))}$$

This completes our description of the proposed scheme. Observer that as with SAOTS, the scheme is fully transparent to the receiver since the signature is a standard signature. The certificate of the user specifying P_U may be supplied along with the signature if required. The receiver does not need to have a custom built software to check the validity of the signature. This is in contrast to SAS. Further, the user is not required to perform any public key operations whereas in SAS, digital signature verification was required by the user.

4.3 Analysis

First observe that for every signature generated, the VS has to add 3 hash values to its storage, i.e., for the ith signature, the VS has to store the following: 1) $h(M_i)$, 2) $h(S_U^i(h(M_i), h(P_U^{i+1})))$, and 3) $h(P_U^i)$. Thus, a signature costs only 60 bytes to the VS. Clearly, this is a significant improvement over [1] in which the VS had to add about 5 KB to its storage as explained in section 3. After generating i signatures for the user U, the VS stores the following-

$$\mathbf{\{h(M_1), h(S_U^1(h(M_1), h(P_U^2))), h(P_U^1)\}, \{h(M_2), h(S_U^2(h(M_2), h(P_U^3))), h(P_U^2)\}, \ldots, \{h(M_i), h(S_U^i(h(M_i), h(P_U^{i+1}))), h(P_U^i)\}, P_U^{i+1}}$$

Now, consider memory requirements for the mobile user. In our scheme, the user is only required to store the 128 bit secret key K along with the counter i. Assuming a 32 bit counter, the memory requirements for the user comes out to be 20 bytes. This is about 135 times less than that in the original scheme [1] for which the requirements are about 2.7 KB as explained previously.

4.4 Dispute Resolution

A dispute arises in case a user U claims that the VS has signed a message M on his behalf which he did not request the VS to sign. The prerequisite for this claim is that U should produce the signature on M generated by the VS on his behalf. Now, the arbiter asks VS to prove that the message M was indeed requested by U to be signed. Additionally, since in our scheme the VS does not store full one time signatures and public keys, the arbiter also asks U to cooperate in the process (although U may not do so honestly).

At this point, assume that VS has generated n signatures on behalf of the user U. Thus, VS stores the following:

$\{h(M_1), h(S_U^1(h(M_1), h(P_U^2))), h(P_U^1)\}, \{h(M_2), h(S_U^2(h(M_2), h(P_U^3))), h(P_U^2)\}, \ldots, \{h(M_n), h(S_U^n(h(M_n), h(P_U^{n+1}))), h(P_U^n)\}, P_U^{n+1}$

The dispute resolution protocol proceeds in n steps. The ith step of the process takes place as follows for $1 \leq i \leq n$

(1) The VS submits the stored $h(S_U^i(h(M_i), h(P_U^{i+1})))$ to the arbiter.

(2) VS demands P_U^i and $S_U^i(h(M_i), h(P_U^{i+1}))$ from U after supplying him $h(M_i)$ and $h(P_U^{i+1})$. U generates S_U^i and P_U^i using the secret key K (see footnote[1]) and signs the concatenation of supplied $h(M_i)$ and $h(P_U^{i+1})$ using S_U^i. U cannot supply a wrong P_U^i since he signed $h(P_U^i)$ in the previous i.e. (i-1)th step[2]. He cannot supply a wrong signature $S_U^i(h(M_i), h(P_U^{i+1}))$ since the signature can be verified using the supplied key P_U^i.

[1] Note that it is immaterial how U generates one time key pairs S_U^i's and P_U^i's. VS cannot and need not ensure that U is following the correct formula for generating one time key pairs during any phase in the scheme. The sole purpose of specifying a formula for S_U^i and P_U^i is to put the responsibility of supplying signatures and public keys on U instead of the VS.

[2] for i = 1, U cannot supply a wrong key since P_U^1 was the initial key registered using non-repudiable means.

(3) The arbiter computes the hash of the signature $h(S_U^i(h(M_i), h(P_U^{i+1})))$ supplied by U and matches it with hash of the signature submitted by VS in (1). If they do not match, the VS is concluded to be the cheater. Otherwise, the arbiter concludes the following-
   a) VS was requested by U to sign $h(M_i)$
   b) Hash of the next one time public key to be supplied by U should be $h(P_U^{i+1})$.

If the hash of the disputed message, i.e., h(M) equals $h(M_i)$, the dispute is resolved in the favour of VS. Else if i equal n, i.e., all message stored by VS have been checked and none equals M, the dispute is resolved in the favour of the user U. Otherwise, i is incremented and the process continues with step (1) again.

Now, we summarize the computational and storage requirements for SAOTS and our scheme for all the 3 parties involved.

Notations:

H: hash computation
S: generation of traditional public key signature
V: verification of traditional public key signature
E: OTS encoding computation (costs less than one hash)
p: number of hash computations to verify OTS
m: number of public key components in the OTS scheme
K: size of secret key of the user
C: size of the signature counter

	Party Name	SAOTS original	Proposed Construction
Computational Requirements	User	1H + 1E	1H + 1E
	Server	1E + (p+2)H + 1S	1E + (p+3)H + 1S
	Receiver	1H + 1V	1H + 1V
Storage Requirements	User	mH	1K + 1C
	Server	(m+p+1)H	3H
	Receiver	_	_

Table 1: Objective comparison of the proposed scheme with the original SAOTS scheme

5 Conclusion

Server assisted signature schemes try to provide a means of non-repudiation for constrained and mobile devices having limited computational and memory resources. SAOTS seems to be the most practical server assisted signature scheme. However, the main problem with SAOTS was high storage requirements for the VS and memory requirements for the mobile client.

We improved the SAOTS construction by addressing both of the above concerns. By generating one time signature keys pseudorandomly and introducing a new server storage and dispute resolution scheme, we are able to substantially reduce the VS storage requirements as well as the mobile client memory requirements. Only 60 bytes per signature generation are added to the VS storage instead of 5 KB in the previous scheme. Further, the memory requirements for the mobile client are reduced from 2.7 KB to just 20 bytes. The resulting construction seems to be practical from both computational as well as storage point of view.

References

[1] Kemal Bicakci and Nazife Baykal, “Server Assisted Signatures Revisited”, T. Okamoto (Ed.): CT-RSA 2004, LNCS 2964, pp. 143–156, 2004.

[2] M. Burnside, D. Clarke, T. Mills, A. Maywah, S. Devadas, and R. Rivest. Proxy- Based Security Protocols in Networked Mobile Devices. Proceedings of the 17th ACM Symposium on Applied Computing (Security Track), March 2002.

[3] A. Boldyreva, A. Palacio, and B. Warinschi. Secure Proxy Signature Schemes for Delegation of Signing Rights. Cryptology ePrint Archive, Report 2003/096, 2003, http://eprint.iacr.org.

[4] P. Beguin and J. J. Quisquater. Fast server-aided RSA signatures secure against active attacks. CRYPTO 95, LNCS No. 963, Springer-Verlag, 1995.

[5] P. Nguyen and J. Stern. The Beguin-Quisquater server-aided RSA protocol from Crypto ’95 is not secure. ASIACRYPT 98, LNCS No. 1514, Springer-Verlag, 1998.

[6] M. Jakobsson and S. Wetzel. Secure Server-Aided Signature Generation. In Proc. of the International Workshop on Practice and Theory in Public Key Cryptography (PKC 2001), LNCS No. 1992, Springer, 2001.

[7] N. Asokan, G. Tsudik and M. Waidners. Server-supported signatures. Journal of Computer Security, November 1997.

[8] X. Ding, D. Mazzocchi and G. Tsudik. Experimenting with Server-Aided Signatures. Network and Distributed Systems Security Symposium (NDSS’02), February 2002.

[9] K. Bicakci and N. Baykal. SAOTS: A New Efficient Server Assisted Signature Scheme for Pervasive Computing. In Proc. of 1st International Conference on Security in Pervasive Computing, SPC 2003, LNCS No. 2802, March 2003, Germany.

[10] K. Bicakci and N. Baykal. Design and Performance Evaluation of a Flexible and Efficient Server Assisted Signature Protocol. In Proc. of IEEE 8th Symposium on Computers and Communications, ISCC 2003, Antalya, Turkey.

[11] R. Gennaro and P. Rohatgi. How to Sign Digital Streams. CRYPTO 1997, LNCS No. 1294, Springer-Verlag, 1997.

[12] National Institute for Standards and Technology. Digital Signature Standard (DSS). Federal Register, 56(169), August 30, 1991.

[13] National Institute of Standards and Technology (NIST), "Announcing the Secure Hash Standard", FIPS 180-1, U.S. Department of Commerce, April 1995.

[14] R. L. Rivest, A. Shamir, and L. M. Adleman, “A method for obtaining digital signatures and public-key cryptosystems”, Communications of the ACM, 21(2):120–126, 1978.

[15] L. Lamport, “Password Authentication with Insecure Communication”, Communications of the ACM 24.11 (November 1981), pp 770-772.

[16] Kan Zhang, Efficient Protocols for Signing Routing Messages, Proceedings of the Network and Distributed System Security Symposium, San Diego, California, USA, 1998.

[17] R.C. Merkle, A Digital Signature Based on a Conventional Encryption Function, Proc. CRYPTO’87, LNCS 293, Springer Verlag, 1987, pp 369-378.

[18] R.C. Merkle, A Certified Digital Signature, Proc. CRYPTO’89, LNCS 435, Springer Verlag, 1990, pp 218-238.

[19] D. Bleichenbacher and U.M. Maurer, Directed Acyclic Graphs, One-way Functions and Digital Signatures, Proc. CRYPTO’94, LNCS 839, Springer Verlag, 1994, pp 75-82.

[20] D. Bleichenbacher, U.M. Maurer, Optimal Tree-Based One-time Digital Signature Schemes, Proc. STACS’96, LNCS 1046, Springer-Verlag, pages: 363-374, 1996.

[21] D. Bleichenbacher, U.M. Maurer, On the efficiency of one-time digital signatures, Proc. ASIACRYPT’96, LNCS 1163. Springer-Verlag, pages: 145-158, 1996.

[22] K. Bicakci, G. Tsudik, B. Tung, How to construct optimal one-time signatures, Computer Networks (Elsevier), Vol.43(3), pp. 339-349, October 2003.